ANSIBLE BEST PRACTICES - WINDOWS

LOGISTICS

- Class Hours:
  - Instructor will provide class start and end times.
  - Breaks throughout class

- Telecommunication:
  - Turn off or set electronic devices to vibrate
  - Reading or attending to devices can be distracting to other students

- Lunch:
  - 1 hour and 15 minutes
  - Extra time for email, phone calls, or simply a walk.

- Miscellaneous
  - Courseware
  - Bathroom

COURSE OBJECTIVES

- Investigate recommended practices for effective and efficient automation with Ansible.
- Perform rolling updates with your Ansible Automation operations.
- Use advanced features of Ansible to work with data, including filters and plugins.
- Implement Red Hat Ansible Controller to coordinate and scale Red Hat Ansible Automation centrally.
- Leverage the capabilities of Red Hat Ansible Controller to manage complex automation workflows.
- More!

TRAINING DILEMMA

HI!

Jason Smith

Cloud Consultant with an Operations background. Focused on cloud-native technologies: automation, containers & orchestration.

LinkedIn
https://www.linkedin.com/in/jruels/

mail
jason@innovationinsoftware.com

github
https://github.com/jruels

Expertise

- Cloud
- Automation
- CICD
- Docker
- Kubernetes

INTRODUCTIONS

- Name
- Job Role
- Your experience with Automation tools
  - Ansible (scale 1 - 5)
  - Ansible Controller (scale 1 - 5)
- Expectations for course (please be specific)

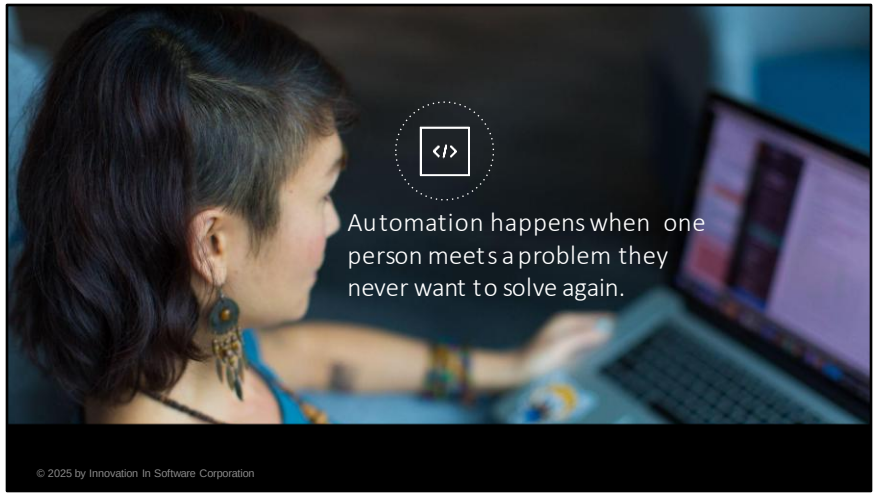
CLASS PAGE

https://jruels.github.io/ansible-windows-best

POP QUIZ: Automation

Why do we automate tasks?

5 MINUTES


Automation happens when one person meets a problem they never want to solve again.
© 2025 by Innovation In Software Corporation

AUTOMATION

Repeatability by automating activities.

AUTOMATION TOOLS

ANSIBLE

CHEF

puppet

HashiCorp Terraform

INFRASTRUCTURE AS CODE

What is IaC?

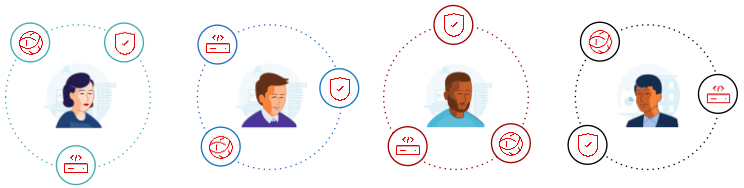
Infrastructure as code (IaC) is the process of managing and provisioning computer data centers through machine-readable definition files
Used with bare-metal as well as virtual machines and many other resources.
Normally a declarative approach

INFRASTRUCTURE AS CODE

- Programmatically provision and configure components
- Treat like any other code base
  - Version control
  - Automated testing
  - data backup

Many organizations share the same challenge

Too many unintegrated, domain-specific tools



Ansible Introduction

ANSIBLE ARCHITECTURE

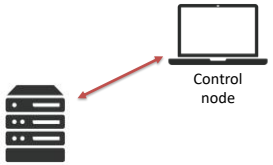
Control
node

ANSIBLE CONTROL NODE

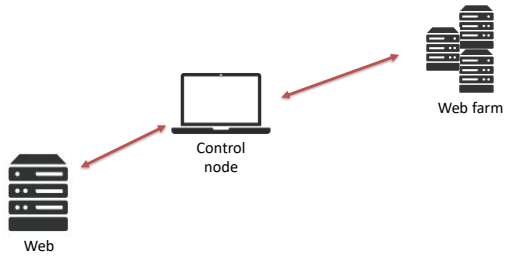
The machine from which you run the Ansible CLI tools (`ansible-playbook` , `ansible`, `ansible-vault` and others).

You can use any computer that meets the software requirements as a control node - laptops, shared desktops, and servers can all run Ansible. Multiple control nodes are possible, but Ansible itself does not coordinate across them, see AAP for such features.

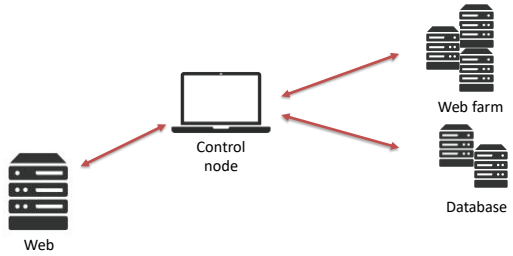
ANSIBLE ARCHITECTURE



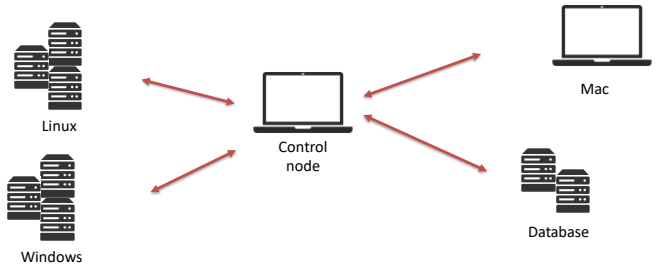
ANSIBLE ARCHITECTURE



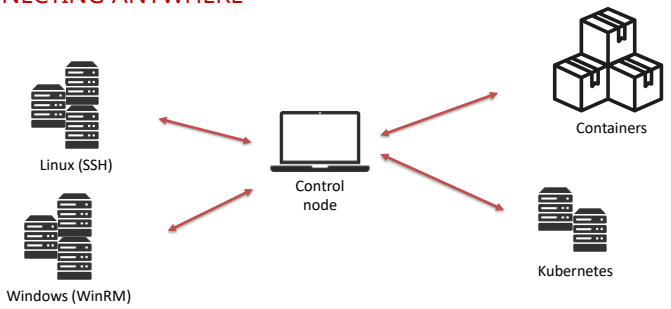
ANSIBLE ARCHITECTURE



CONNECTING ANYWHERE



CONNECTING ANYWHERE



MANAGED NODE

Also referred to as ‘hosts’, these are the target devices (servers, network appliances or any computer) you aim to manage with Ansible.

Ansible is not normally installed on managed nodes, unless you are using `ansible-pull`, but this is rare and not the recommended setup.

POP QUIZ: DISCUSSION

When does it make sense to use `ansible-pull`?

POP QUIZ: DISCUSSION

When does it make sense to use `ansible-pull`?

- Scaling

POP QUIZ: DISCUSSION

When does it make sense to use `ansible-pull`?

- Scaling
- Periodic remediation

POP QUIZ: DISCUSSION

When does it make sense to use `ansible-pull`?

- Scaling
- Periodic remediation
- Similar to Chef & Puppet

INVENTORY

Ansible works against multiple managed nodes or “hosts” in your infrastructure at the same time, using a list or group of lists known as inventory.

Once your inventory is defined, you use patterns to select the hosts or groups you want Ansible to run against.

INVENTORY

The default location for inventory is a file called

- `/etc/ansible/hosts`

You can specify another inventory file/directory at the command-line using the `-i <path>` option.

You can also use multiple inventory files at the same time and/or pull inventory from dynamic or cloud sources.

INVENTORY

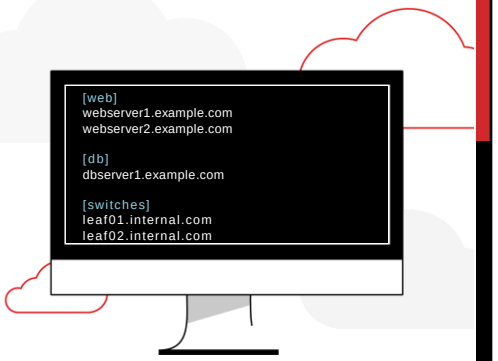
- Ansible works against multiple systems in an inventory
- Inventory is usually file based
- Can have multiple groups
- Can have variables for each group or even host

ANSIBLE INVENTORY

The systems that a playbook runs against

What are they?

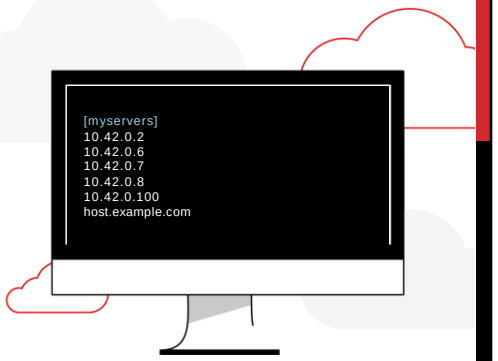
List of systems in your infrastructure that automation is executed against



ANSIBLE INVENTORY

The Basics

An example of a static Ansible inventory including systems with IP addresses as well as fully qualified domain name (FQDN)



INVENTORY VARIABLES


```
[app1srv]
appserver01 ansible_host=10.42.0.2
appserver02 ansible_host=10.42.0.3

[web]
node-[1:30] ansible_host=10.42.0.[31:60]

[web:vars]
apache_listen_port=8080
apache_root_path=/var/www/mywebdocs/

[all:vars]
ansible_user=kev
ansible_ssh_private_key_file=/home/kev/.ssh/id_rsa
```

ANSIBLE INVENTORY FORMATS

Ansible inventory can be expressed in `'INI'` or `'YAML'` format.

Example (INI):

```
mail.example.com

[webservers]
foo.example.com
bar.example.com

[dbservers]
one.example.com
two.example.com
three.example.com
```


The headings in brackets are group names, which are used in classifying hosts and deciding what hosts you are controlling at what times and for what purpose

ANSIBLE INVENTORY FORMATS

Here's the same basic inventory file in YAML format:

Example (YAML):

```
all:
  hosts:
    mail.example.com:
  children:
    webservers:
      hosts:
        foo.example.com:
        bar.example.com:
    dbservers:
      hosts:
        one.example.com:
        two.example.com:
        three.example.com:
```


The headings in brackets are group names, which are used in classifying hosts and deciding what hosts you are controlling at what times and for what purpose

INVENTORY GROUPS

Ansible Inventory - The Basics

```
[nashville]
appserver01 ansible_host=10.42.0.2
appserver02  ansible_host=10.42.0.3

[atlanta]
node-[1:30]  ansible_host=10.42.0.[31:60]

[south:children]
Atlanta
Nashville
hsvapp05
```

37

37

The headings in brackets are group names, which are used in classifying hosts and deciding what hosts you are controlling at what times and for what purpose

ANSIBLE INVENTORY GROUPS

There are two default groups:

- all: Contains every host
- ungrouped: Contains all hosts that don't have another group aside from all.

Every host will always belong to at least 2 groups (all and ungrouped or all and some other group).

Though all and ungrouped are always present, they can be implicit and not appear in group listings.

The headings in brackets are group names, which are used in classifying hosts and deciding what hosts you are controlling at what times and for what purpose

ANSIBLE INVENTORY GROUPS

You can (and probably will) put each host in more than one group. For example, a production webserver in a datacenter in Atlanta might look like this:

```
all:
  children:
    prod:
      hosts:
        web1.example.com:
    atlanta:
      hosts:
        web1.example.com
    webservers:
      hosts:
        web1.example.com
```


[prod] and [atlanta] and [webservers]. You can create groups that track:
What - An application, stack or microservice (for example, database servers, web servers, and so on).
Where - A datacenter or region, to talk to local DNS, storage, and so on (for example, east, west).
When - The development stage, to avoid testing on production resources (for example, prod, test).

ANSIBLE INVENTORY RANGES

If you have a lot of hosts with a similar pattern, you can add them as a range rather than listing each hostname separately:
Example (INI):

```
[webservers]
www[01:50].example.com
...
  webservers:
    hosts:
      www[01:50].example.com:
```


If you have a lot of hosts with a similar pattern, you can add them as a range rather than listing each hostname separately:

ANSIBLE DYNAMIC INVENTORY

If your Ansible inventory fluctuates over time, with hosts spinning up and shutting down in response to business demands you may need to track hosts from multiple sources: cloud providers, LDAP, Cobbler, and/or enterprise CMDB systems.

Ansible integrates all of these options through a dynamic inventory system. Ansible uses inventory plugins to keep track of managed hosts.

If you have a lot of hosts with a similar pattern, you can add them as a range rather than listing each hostname separately:

ANSIBLE DYNAMIC INVENTORY

Dynamic inventory supports many solutions including Cloud Providers.
Example (AWS):

```
plugin: aws_ec2
regions:
  - us-west-1
filters:
  instance-state-name: running
keyed_groups:
  - key: tags['role']
    prefix: tag_role
hostnames:
  - ip-address
```


If you have a lot of hosts with a similar pattern, you can add them as a range rather than listing each hostname separately:

POP QUIZ: DISCUSSION

Where is the default inventory location?

POP QUIZ: DISCUSSION

Where is the default inventory location?

- `/etc/ansible/hosts`

POP QUIZ: DISCUSSION

Which formats are valid for an Ansible inventory?

POP QUIZ: DISCUSSION

Which formats are valid for an Ansible inventory?

- INI
- YAML

ANSIBLE AD-HOC

An Ansible ad hoc command uses the `/usr/bin/ansible` command-line tool to automate a single task on one or more managed nodes. ad hoc commands are quick and easy, but they are not reusable.

Why learn about ad hoc commands first? ad hoc commands demonstrate the simplicity and power of Ansible. The concepts you learn will port over directly to the playbook language.

POP QUIZ: DISCUSSION

What are some use-cases for ad-hoc mode?

POP QUIZ: DISCUSSION

What are some use-cases for ad-hoc mode?

- Copy files

POP QUIZ: DISCUSSION

What are some use-cases for ad-hoc mode?

- Copy files
- Manage packages, users, groups

POP QUIZ: DISCUSSION

What are some use-cases for ad-hoc mode?

- Copy files
- Manage packages, users, groups
- Reboot servers

ANSIBLE AD-HOC

Scripted Ad-hoc

```
ansible all -m win_copy -a "src=master.gitconfig dest=C:\<User>\.gitconfig"

ansible -m win_chocolatey -a "name=git state=present" localhost

ansible -i hosts all -m win_ping -u Administrator
```

ANSIBLE AD-HOC

ad hoc commands are great for tasks you repeat rarely. For example, if you want to power off all the machines in your lab for Christmas vacation, you could execute a quick one-liner in Ansible without writing a playbook. An ad hoc command looks like this:

```
$ ansible [pattern] -m [module] -a "[module options]"
```

Example of installing packages on nodes in webservers group.

```
$ ansible win_servers -m win_feature -a "name=Web Server state=present"
```


Notice how the "become" key is lined up with the "block". This means it is applied to the entire block.

YAML

Playbooks are written in YAML.
YAML is used because it is easier for humans to read and write than other data formats like XML and JSON.
It is a format widely used by many tools (Kubernetes, Docker compose, Machine Learning, etc.)

YAML BASICS

For Ansible, nearly every YAML file starts with a list. Each item in the list is a list of key/value pairs, commonly called a “hash” or a “dictionary”. So, we need to know how to write lists and dictionaries in YAML.

There’s another small quirk to YAML. All YAML can optionally begin with --- and end with .... This is part of the YAML format and indicates the start and end of a document.

YAML BASICS

All members of a list are lines beginning at the same indentation level starting with a "- " (a dash and a space):

Example:

```
---
# A list of tasty fruits
- Apple
- Orange
- Strawberry
- Mango
...
```

YAML BASICS

A dictionary is represented in a simple key: value form (the colon must be followed by a space):

Example:

```
---
# An employee record
martin:
  name: Martin D'vloper
  job: Developer
  skill: Elite
```

YAML BASICS

More complicated data structures are possible, such as lists of dictionaries, dictionaries whose values are lists or a mix of both:

Example:

```
---
# Employee records
- martin:
    name: Martin D'vloper
    job: Developer
    skills:
      - python
      - perl
      - pascal
- tabitha:
    name: Tabitha Bitumen
    job: Developer
    skills:
      - lisp
      - fortran
      - erlang
```

YAML BASICS

Values can span multiple lines using | or >.

Spanning multiple lines using a “Literal Block Scalar” | will include the newlines and any trailing spaces.

Using a “Folded Block Scalar” > will fold newlines to spaces; it’s used to make what would otherwise be a very long line easier to read and edit.

In either case the indentation will be ignored.

YAML BASICS

Example:

```
include_newlines: |
        exactly as you see
        will appear these three
        lines of poetry

fold_newlines: >
        this is really a
        single line of text
        despite appearances
```

ANSIBLE PLAYBOOK

Ansible Playbooks offer a repeatable, reusable, simple configuration management and multi-machine deployment system, one that is well suited to deploying complex applications.

If you need to execute a task with Ansible more than once

- Write a playbook
- Put it under source control.

Then you can use the playbook to push out new configurations or confirm the configuration of remote systems.

ANSIBLE PLAYBOOK

Playbooks can:

- Declare configurations
- Orchestrate steps of any manual ordered process, on multiple sets of machines, in a defined order
- Launch tasks synchronously or asynchronously

ANSIBLE PLAYBOOK

A playbook is composed of one or more ‘plays’ in an ordered list.

The terms ‘playbook’ and ‘play’ are sports analogies. Each play executes part of the overall goal of the playbook, running one or more tasks. Each task calls an Ansible module.

PLAYBOOK

YAML

playbook.yml

```
hosts: localhost
tasks :
-   win_copy :
        src:   "master.gitconfig"
        dest:  "C:\Users\Administrator\.gitconfig"
```

PLAYBOOK

YAML

playbook.yml

```
hosts: localhost
tasks :
-   copy :
        src:   "master.gitconfig"
        dest:  "C:\Users\Administrator\.gitconfig"
```

>_

```
ansible-playbook playbook.yml
```

ANSIBLE PLAYBOOK EXECUTION

A playbook runs in order from top to bottom. Within each play, tasks also run in order from top to bottom.

Playbooks with multiple 'plays' can orchestrate multi-machine deployments, running one play on your webservers, then another play on your database servers, then the third play on your network infrastructure, and so on.

ANSIBLE PLAYBOOK TIPS

At a minimum, each play defines two things:

- The managed nodes to target, using a pattern
- At least one task to execute

- Ansible creates a <playbook>.retry playbook for hosts where it failed. You can execute the <playbook>.retry playbook and it will try to run it ONLY on the hosts that failed.
- Limit: Used to run Ansible playbook only on hosts you specify. Great for testing on one host.
- Whitespace: Ansible is like Python, it requires correct indentation. (ansible lint, syntax-check, etc.)

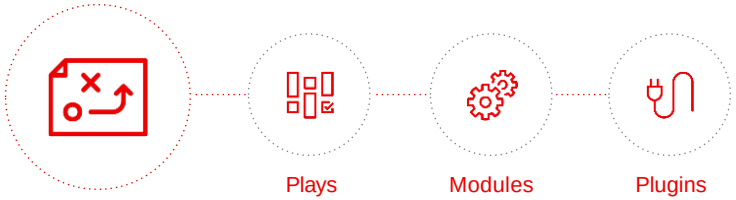
PLAYBOOKS

Simple playbook to install and configure IIS web server

```
---
- name: install and start IIS
  hosts: web

  tasks:
    -  name: IIS package is present
       win_feature:
         name: "web-server"
         state: present
         restart: yes

    - name: latest index.html file is present
      win_copy:
        content: "Hello World"
        dest: C:\Inetpub\wwwroot\index.html

    - name: iis is started
      win_service:
        name: W3Svc
        state: started
```

PLAYBOOKS

What makes up an Ansible playbook?



ANSIBLE PLAYS

What am I automating?

What are they?

Top level specification for a group of tasks. Will tell that play which hosts it will execute on and control behavior such as fact gathering or privilege level.

Building blocks for playbooks

Multiple plays can exist within an Ansible playbook that execute on different hosts.

```
---
- name: install IIS
  hosts: web
  become: yes
```

ANSIBLE MODULES

The “tools in the toolkit”

What are they?

Parametrized components with internal logic, representing a single step to be done.

The modules “do” things in Ansible.

Language

Usually Python, or Powershell for Windows setups. But can be of any language.

```
- name: latest index.html file
  src: files/index.html
  dest: C:\Inetpub\wwwroot\index.html
```

ANSIBLE PLUGINS

The “extra bits”

What are they?

Plugins are pieces of code that augment Ansible’s core functionality. Ansible uses a plugin architecture to enable a rich, flexible, and expandable feature set.

```
---
- name: install and start IIS
  hosts: web
```

Example filter plugins:

```
{{ some_variable | to_nice_json }}
{{ some_variable | to_nice_yaml }}
```

MORE COMPLETE PLAYBOOK

ansible_complete_vars.yml

```yaml
---
- hosts: windows
  name: This is a play within a playbook
  vars:
    iis_sites:
      - name: 'Ansible Playbook Test'
        port: '8080'
        path: 'C:\sites\playbooktest'
      - name: 'Ansible Playbook Test 2'
        port: '8081'
        path: 'C:\sites\playbooktest2'
    iis_test_message: "Hello World!  My test IIS
Server"
```

MORE COMPLETE PLAYBOOK

ansible_complete_part1.yml

```yaml
tasks:
    - name: Install IIS
      win_feature:
        name: Web-Server
        state: present

    - name: Create site directory structure
      win_file:
        path: "{{ item.path }}"
        state: directory
      with_items: "{{ iis_sites }}"
```

MORE COMPLETE PLAYBOOK

ansible_complete_part2.yml

```yaml
- name: Create IIS site
  win_iis_website:
    name: "{{ item.name }}"
    state: started
    port: "{{ item.port }}"
    physical_path: "{{ item.path }}"
  with_items: "{{ iis_sites }}"
  notify: restart iis service

- name: Open port for site on the firewall
  win_firewall_rule:
    name: "iisport{{ item.port }}"
    enable: yes
    state: present
    localport: "{{ item.port }}"
    action: Allow
    direction: In
    protocol: Tcp
  with_items: "{{ iis_sites }}"
```

MORE COMPLETE PLAYBOOK

ansible_complete_part3.yml

```yaml
  - name: Template simple web site to iis_site_path as index.html
    win_template:
      src: 'index.html.j2'
      dest: '{{ item.path }}\index.html'
    with_items: "{{ iis_sites }}"

  - name: Show website addresses
    debug:
      msg: "{{ item }}"
    loop:
      - http://{{ ansible_host }}:8080
      - http://{{ ansible_host }}:8081

handlers:
  - name: restart iis service
    win_service:
      name: W3Svc
      state: restarted
      start_mode: auto
```

ANSIBLE VARIABLES

Ansible uses variables to manage differences between systems. With Ansible, you can execute tasks and playbooks on multiple different systems with a single command.

- Create variables with YAML syntax, including lists and dictionaries
- Define these variables

ANSIBLE VARIABLES

Variables can be defined in many locations:

- Playbooks
- Inventory
- re-usable files or roles
- command line.

You can also create variables during a playbook run by registering the return value or values of a task as a new variable.

ANSIBLE VARIABLE VALID NAMES

A variable name can only contain:

- Letters
- Numbers
- Underscores

A variable name cannot begin with a number but can begin with an underscore.

POP QUIZ: DISCUSSION

Are these valid variable names?

- foo

POP QUIZ: DISCUSSION

Are these valid variable names?

- foo - valid

POP QUIZ: DISCUSSION

Are these valid variable names?

- app.port

POP QUIZ: DISCUSSION

Are these valid variable names?

- app.port - invalid

POP QUIZ: DISCUSSION

Are these valid variable names?

- *ssl_key

POP QUIZ: DISCUSSION

Are these valid variable names?

- *ssl_key - invalid

POP QUIZ: DISCUSSION

Are these valid variable names?

- _web_root

POP QUIZ: DISCUSSION

Are these valid variable names?

- _web_root - valid

SIMPLE VARIABLE

Defining simple variables

```
remote_install_path: C:\opt\my_app_config
```

After you define a variable, use Jinja2 syntax to reference it. Jinja2 variables use double curly braces.

```
ansible.builtin.win_template:
  src: foo.cfg.j2
  dest: '{{ remote_install_path }}\foo.cfg'
```


Notice how the "become" key is lined up with the "block". This means it is applied to the entire block.

VARIABLE QUOTATION

If you start a value with {{ foo }}, you must quote the whole expression to create valid YAML syntax. If you do not quote the whole expression, the YAML parser cannot interpret the syntax - it might be a variable or it might be the start of a YAML dictionary.

This example will error.

```
- hosts: app_servers
  vars:
    app_path: {{ base_path }}/22
```

ERROR! Syntax Error while loading YAML

VARIABLE QUOTATION

If you start a value with {{ foo }}, you must quote the whole expression to create valid YAML syntax. If you do not quote the whole expression, the YAML parser cannot interpret the syntax - it might be a variable or it might be the start of a YAML dictionary.

Fix the issue by quoting the entire expression.

```
- hosts: app_servers
  vars:
    app_path: "{{ base_path }}\22"
```

ANSIBLE MODULES

Modules are the main building blocks of Ansible playbooks. Although we do not generally speak of "module plugins", a module is a type of plugin.

Common modules:

- Working with files: copy, archive, unarchive, get_url
- user, group
- ping
- service
- win_feature, win_package, chocolatey
- win_template

ANSIBLE MODULES

Common modules:

- win_lineinfile
  - Manipulate text in files
    - Add alias for hosts
    - Supports regex
    - Idempotent
- win_shell/win_command
- win_powershell
- debug module